Progetto:

EvenTrento

Titolo del documento:

Implementazione

Autore:

Togni Roberto

Document Info:

Doc. Name	D3-EvenTrentoImplementazione	Doc. Number	D3 V0.1
Description	Documento di descizione dell'implementazione		

Dipartimento di Ingegneria e Scienza dell'Informazione

Contents

1 Scopo del Documento

Il presente documento ha lo scopo di riportare le informazioni riguardanti lo sviluppo di una parte dell'applicazione EvenTrento. Partendo dalle user stories, il documento presenta un walk-throgh delle funzionalità effettivemente implementate, con un focus sull'interazione dell'utente con queste ultime. Prosegue dunque con la descrizione delle web APIs del'applicazione, nonché della modalità di sviluppo collaborativo, del database e del testing delle varie funzionalità. Infine, viene presentato il front-end realizzato, accompagnato da una descrizione della modalità di deployment utilizzata.

2 User Stories

Le seguenti tabelle raccolgono una sere di User Stories divise in diverse epiche. Ad ogni user story è associata una priorità (ad un numero più piccolo corrisponde una priorità maggiore) rappresentante l'importanza della funzionalità in questione per l'operatività del sistema.

Profilo			
Id	Nome	User Story	Priorità
A1	Registrazione	Come utente, voglio registrarmi per utilizzare il sistema	1
A2	Login	In qualità di utente, devo essere in grado di effettuare il login	1
A3	Visualizzazione profilo	Come utente, devo essere in grado di visualizzare lo storico eventi, le mie informazioni personali, e gli eventi salvati	2

Table 1: User stories relative all'epica "profilo".

Eventi			
Id	Nome	User Story	Priorità
B1	Lista eventi	In qualità di utente, devo essere in grado di scorrere una lista di eventi	2
B2	Filtri eventi	In qualità di utente, devo essere in grado di filtrare gli eventi in base alla data	2
B3	Creazione evento	In qualità di organizzatore, devo essere in grado di creare un nuovo evento in un determinato luogo	2
B4	Condivisione evento	In qualità di utente, devo essere in grado di condividere un evento tramite link	4
B5	Pagamento	In qualità di utente, devo essere in grado di effettuare il pagamento per un evento a cui voglio iscrivermi	2
B6	Aggiornamento evento	In qualità di organizzatore, devo essere in grado di aggiornare la descrizione di un evento	3
B7	Statistiche evento	In qualità di organizzatore, devo essere in grado di visualizzare gli iscritti all'evento	4
B8	Salvataggio evento	In qualità di utente, devo essere in grado di salvare un evento in modo da poterlo visualizzare nella mia area personale	3

Table 2: User stories relative all'epica "eventi".

Spazi			
Id	Nome	User Story	Priorità
C1	Aggiunta spazio	In qualità di owner, devo essere in grado di pubblicare uno spazio disponibile	2
C2	Rimozione spazio	In qualità di owner, devo essere in grado di rimuovere uno spazio esistente	2
C3	Visualizzazione spazi	In qualità di organizzatore, devo essere in grado di visualizzare gli spazi disponibili	3

Table 3: User stories relative all'epica "spazi".

Mappa			
Id	Nome	User Story	Priorità
D1	Esplorazione mappa	In qualità di utente, devo essere in grado di muovermi nella mappa per visualizzare gli eventi	1

Table 4: User stories relative all'epica "mappa".

3 User Flow

Di seguito sono riportati i diagrammi di flusso rappresentanti gli user flow corrispondenti ad una serie di user stories. La notazione utilizzata nelle descrizioni si riferisce a quella riportata nelle Tabelle 1, 2, 3 e 4.

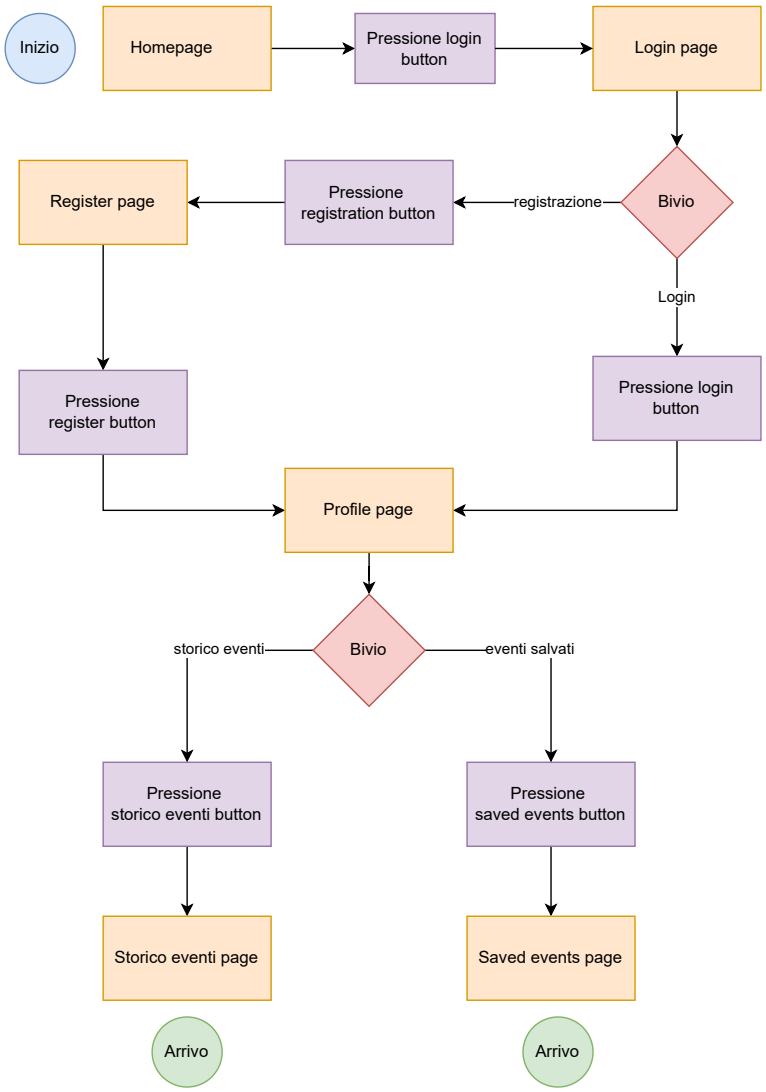

Figure 1: User Flowchart relativo alle user stories A1, A2 e A3.

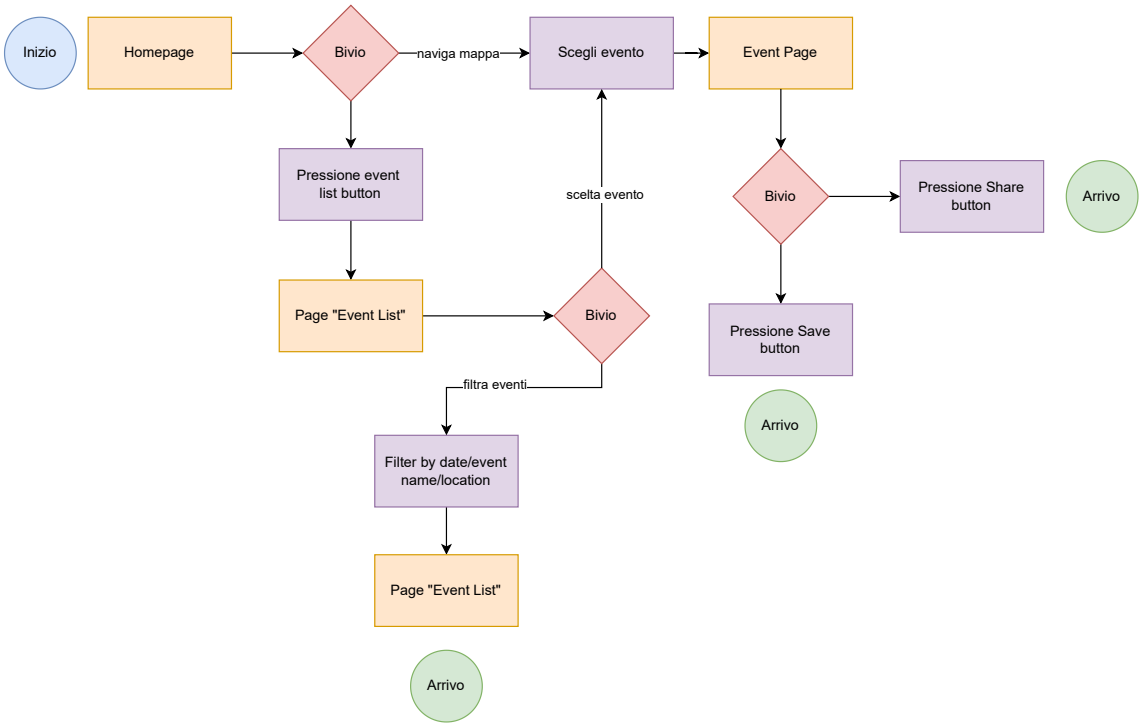

Figure 2: User Flowchart relativo alle user stories B1, B2, B4, B8 e D1.

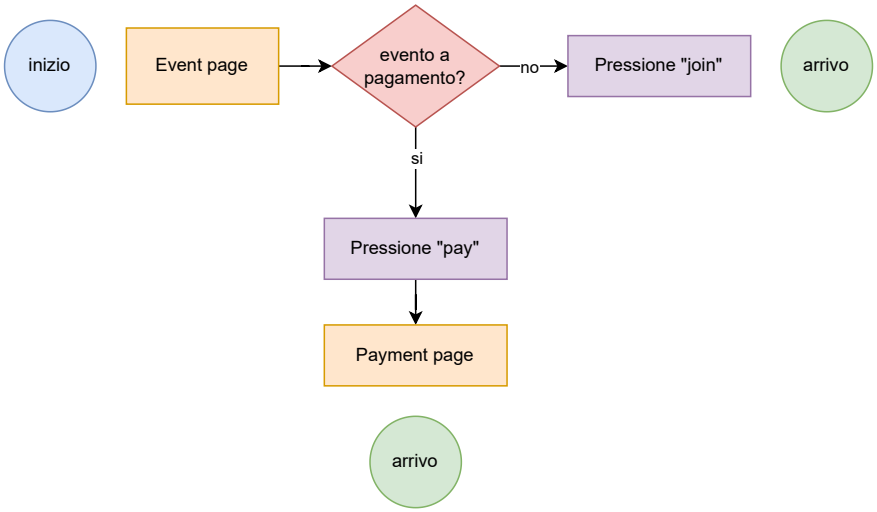

Figure 3: User Flowchart relativo alla user story B5.

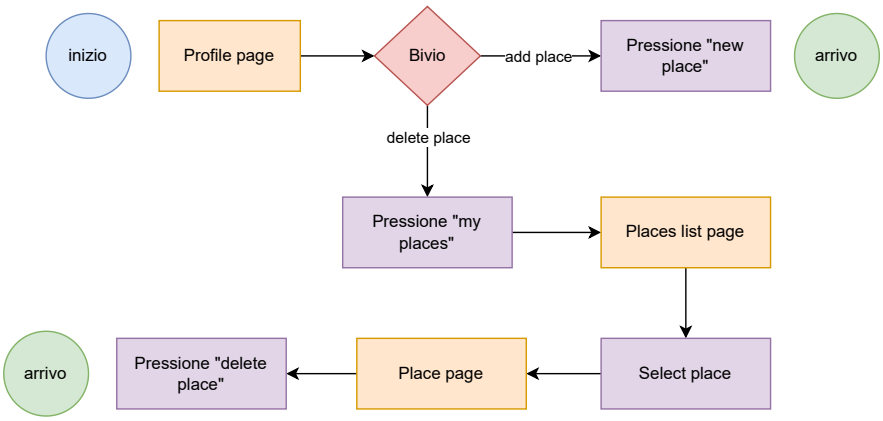

Figure 4: User Flowchart relativo alle user stories C1 e C2.

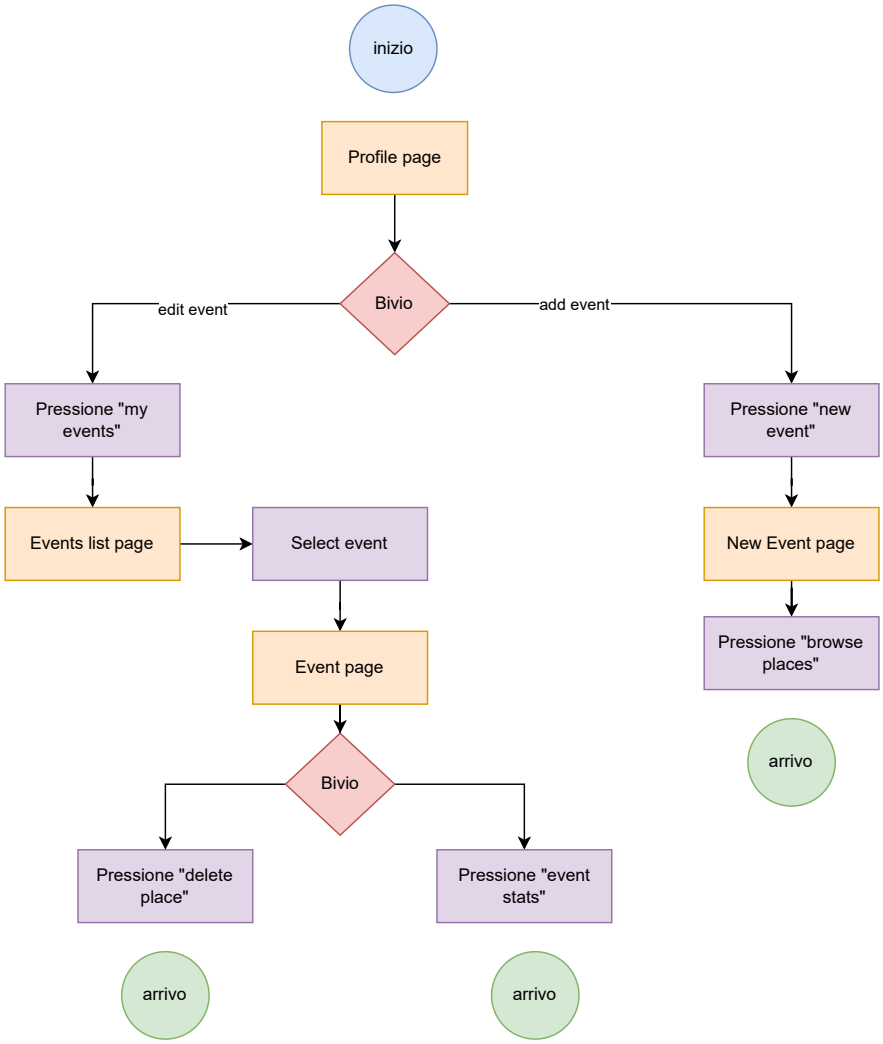

Figure 5: User Flowchart relativo alle user stories B3, B6, B7 e C3.

4 Web APIs

Le API sono state documentate rispettando le specifiche openapi3. La specifica è disponibile nella repository GitHub, ed è per comodità riportata di seguito.

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: "EvenTrento OpenAPI 3.0"
  description: API for managing events.
  license:
  name: MIT
servers:

  - description: Localhost
  url: http://localhost:8000/api/v1

###################
# API ENDPOINTS
###################

paths:
# path per gestire la registrazione di un utente
/users/signup:
  post:
    description: >-
      Creates a new user in the system.
    summary: Register a new user
    requestBody:
    required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/User'
    responses:
      "201":
        description: "User created. Link in the Location header"
        headers:
          "Location":
            schema:
              type: string
              description: Link to the newly created student.
      "400":
        description: "Invalid input. Please verify the request body."
      "409":
        description: "User already exists."

# path per gestire il login di un utente gia' registrato
/users/login:
  post:
    summary: Login to the system
    description: Authenticate the user and return a JWT token.
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              email:
                type: string
                description: "Email address of the user"
              password:
                type: string
                description: "Password for the user account"
            required:
              - email
              - password
    responses:
      '200':
        description: Login successful
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  description: "JWT token to authenticate future requests"
      '400':
        description: Invalid email or password
      '500':
        description: Internal server error
```

```
  # path per gestire gli eventi salvati dall'utente
  /users/saved-events:
    post:
      summary: Save an event of interest for a user
      description: Allows a user to save an event they are interested in.
      security:
        - bearerAuth: []   # Richiede un token JWT valido per accedere
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                eventId:
                  type: integer
                  description: The ID of the event to save
                userId:
                  type: integer
                  description: The ID of the user saving the event
      responses:
        '201':
          description: 'Event saved successfully'
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
      '404':
        description: 'Event not found'
      '400':
        description: 'Invalid save request'
      '500':
        description: 'Internal server error'
    get:
      summary: Retrieve saved events for the authenticated user
      description: Returns a list of events saved by the authenticated user.
      security:
        - bearerAuth: []
      responses:
        '200':
          description: 'List of saved events'
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Event'
      '401':
        description: 'Unauthorized user'
      '500':
        description: 'Internal server error'

  # path per creare e fare il retrieving di eventi
  /events:
    get:
      description: Returns a list of events, optionally filtered by date.
      summary: View all events (with optional date filter)
      parameters:
        - in: query
          name: startDate
          schema:
            type: string
            format: date-time
          required: false
      responses:
        '200':
          description: 'Collection of events'
          content:
            application/json:
              schema:
                type: array
                items:
                $ref: '#/components/schemas/Event'
      '400':
        description: 'Invalid date format'
      '500':
        description: 'Internal server error'


  post:
    description: >-
      Creates a new event in the system.
    summary: Create a new event
    security:
```

```
    - bearerAuth: []   # Richiede un token JWT valido per accedere
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/EventCreation' # Creating a new event
  responses:
    '201':
      description: 'Event successfully created'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Event'  # Returning the created event
  '400':
    description: 'Invalid input, check the event details'
  '403':
    description: 'User not authorized to create an event (only organizers)'
  '500':
    description: 'Internal server error'

# path per la modifica di un evento esistente
/events/{eventId}:
  put:
    summary: Update an existing event
    description: Allows an organizer to update an event they created.
    security:
      - bearerAuth: []   # Richiede un token JWT valido per accedere
    parameters:
      - in: path
        name: eventId
        schema:
          type: integer
        required: true
        description: The ID of the event to be updated
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/EventUpdate'  # Schema per aggiornamento
    responses:
      '200':
        description: 'Event updated successfully'
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Event'
    '403':
      description: 'Forbidden. Organizer is not authorized to modify this event.'
    '404':
      description: 'Event not found'
    '500':
      description: 'Internal server error'

# path per l'iscrizione ad un evento
/events/{eventId}/registration:
  post:
    summary: Register a participant for an event
    description: Allows a participant to register for a specific event.
    security:
      - bearerAuth: []   # Richiede un token JWT valido per accedere
    parameters:
      - in: path
        name: eventId
        required: true
        schema:
          type: integer
    responses:
      '201':
        description: 'Registration successful'
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
    '404':
      description: 'Event not found'
    '400':
      description: 'Invalid registration request'
    '500':
      description: 'Internal server error'

# path per gli spazi
/places:
```

```
    get:
      summary: Retrieve a list of spaces
      description: Returns a list of all available spaces.
      security:
        - bearerAuth: []   # Richiede un token JWT valido per accedere
      responses:
        '200':
          description: 'A list of spaces'
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Place'
        '500':
          description: 'Internal server error'

    post:
      summary: Create a new space
      description: Allows an owner to create a new space for events.
      security:
        - bearerAuth: []   # Richiede un token JWT valido per accedere
      requestBody:
        required: true
        content:
          application/json:
          schema:
            $ref: '#/components/schemas/Place'
      responses:
        '201':
          description: 'Place created successfully'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Place'
      '400':
        description: 'Invalid input'
      '500':
        description: 'Internal server error'

    put:
      summary: Update an existing space
      description: Allows an owner to update a space they created.
      security:
        - bearerAuth: []   # Richiede un token JWT valido per accedere
      parameters:
        - in: path
          name: spaceId
          required: true
          schema:
            type: integer
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/PlaceUpdate'
      responses:
        '200':
        description: 'Place updated successfully'
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Place'
      '403':
        description: 'Forbidden. Owner is not authorized to modify this space.'
      '404':
        description: 'Place not found'
      '400':
        description: 'Invalid input'
      '500':
        description: 'Internal server error'


  ########################
  # COMPONENTS SECTION
  ########################

  components:
    schemas:
      User:
        type: object
        required:
          - username
          - email
          - password
          - role
```

```
        properties:
          username:
            type: string
            description: "Username of the user"
          email:
            type: string
            description: "Email address of the user"
          password:
            type: string
            description: "Password for the user account"
          role:
            type: string
            enum: [basicUser, owner, organizer]
            description: "Role of the user in the system"

      Event:
        type: object
        required:
          - id
          - name
          - date
          - location
          - organizerId
        properties:
          id:
            type: integer
            description: "Unique identifier of the event"
          name:
            type: string
            description: "Name of the event"
          description:
            type: string
            description: "Brief description of the event"
          date:
            type: string
            format: date-time
            description: "Date and time of the event"
          location:
            type: string
            description: "Location where the event will be held"
          organizerId:
            type: integer
            description: "The ID of the organizer who created the event"

      EventCreation:
        type: object
        required:
          - name
          - date
          - location
        properties:
          name:
            type: string
            description: "Name of the event"
          description:
            type: string
            description: "Brief description of the event"
          date:
            type: string
            format: date-time
            description: "Date and time of the event"
          location:
            type: string
            description: "Location where the event will be held"

      EventUpdate:
        type: object
        required:
        - name
        - description
        - date
        - location
        properties:
          name:
            type: string
            description: "Updated name of the event"
          description:
            type: string
            description: "Updated description of the event"
          date:
            type: string
            format: date-time
            description: "Updated date and time of the event"
          location:
            type: string
            description: "Updated location of the event"
```

```
    Place: # oggetto per la creazione e la visualizzazione dello spazio
      type: object
      properties:
        id:
          type: integer
          description: Unique identifier of the space
        name:
          type: string
          description: Name of the space
        location:
          type: string
          description: Location of the space
        capacity:
          type: integer
          description: Maximum capacity of the space

        PlaceUpdate: # oggetto per l'aggiornamento dello spazio
      type: object
      properties:
        name:
          type: string
          description: Updated name of the space
        location:
          type: string
          description: Updated location of the space
        capacity:
          type: integer
          description: Updated maximum capacity of the space


  # modalita' di autenticazione che le API utilizzano
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

Listing 1: Documentazione API

5 Implementation

5.1 Repository Implementation

Il codice del progetto, disponibile su GitHub, è organizzato secondo questo schema:

```
.
+-- D1
+-- D2
+-- D3
+-- D4
+-- front_end
|   +-- index.html
|   +-- pages
|   +-- scripts
|   +-- styles
+-- node_app
|   +-- app.js
|   +-- helperFunctions // funzioni ausiliarie usate durante lo sviluppo
|   +-- .env.example // esempio di file di configurazione delle
|   |                // variabili d'ambiente
|   +-- index.js
|   +-- models // modelli dati mongoose
|   +-- package.json  // file di configurazione del progetto npm
|   +-- package-lock.json
|   +-- routes  // endpoint routes per le APIs
|   +-- tokenChecker.js // middleware per l'autenticazione
+-- README.md
|   +-- .gitignore // configurazione git repo
+-- swagger
    +-- evenTrentoAPIs.yaml // documentazione API
```

Listing 2: Struttura della repository

5.2 Branching strategy e organizzazione del lavoro

Malgrado i propositi di collaborazione iniziali, verso la fine del mese di settembre ho preso la decisione di proseguire lo sviluppo in modo individuale. Come facilmente osservabile dalla commit history del progetto, il contributo degli altri due membri del gruppo è limitato al processo di brainstorming e alla realizzazione di alcuni grafici relativi allo User Flow. Tali grafici non sono presenti nella versione finale del presente documento a causa della loro modifica nel corso dello sviluppo dell'applicazione. Una descrizione dettagliata delle statistiche finali è riportata in Figura **TODO**.

```
git log --format='%aN <%aE>' | sort | uniq -c | sort -nr
```

5.3 Dependencies

Il progetto npm fa uso dei seguenti moduli:

- **Cors** per il supporto alle chiamate cross-origin
- **Express** come framework per il backend
- **Jsonwebtoken** per gestire l'autenticazione
- **Mongoose** per interfacciarsi con mongoDB
- **Axios** per interagire con l'API di OpenStreetMap

Sono inoltre riportate le seguenti dipendenze di sviluppo:

- **Dotenv** per la gestione delle variabili d'ambiente
- **TODO**

5.4 Database

La gestione dei dati è stata realizzata basandosi su un database non relazionale. In particolare, si è fatto uso del servizio offerto da Atlas MongoDB.

Per la gestione dei dati necessari per il funzionamento dell'applicazione sono state definite quattro strutture dati mediante Mongoose:

- Event: per gestire tutte le funzionalità legate alla creazione, modifica ed eliminazione degli eventi
- EventRegistration: per gestire correttamente l'iscrizione e la disiscrizione degli utenti dagli eventi
- Place: per gestire tutte le funzionalità legate alla creazione, modifica ed elimazione dei luoghi
- User: per gestire le tre tipologie di utente e le attività ad esse collegate

5.5 Testing

Le API sono state testate tramite una test-suite sviluppata mediante l'utilizzo di Jest. L'implementazione dei test è organizzata in file `.test.js` situati all'interno della cartella `tests`. Ogni file contiene tutti i test case relativi ad una determinata route.

#	Descrizione	Test Data	Precondizioni	Risultato Atteso	Risultato Effettivo
1	Fetching di tutti gli eventi			Viene ritornato un array contenente tutti gli eventi presenti nel database	Risultato atteso
2	Fetching degli eventi filtrando per nome	Titolo dell'evento		Viene ritornato un array contenente gli eventi che corrispondono con i filtri di ricerca (possibilmente vuoto)	Risultato atteso
3	Fetching degli eventi filtrando per location	Luogo dell'evento		Viene ritornato un array contenente gli eventi che corrispondono con i filtri di ricerca (possibilmente vuoto)	Risultato atteso
4	Fetching degli eventi filtrando per data	Data dell'evento		Viene ritornato un array contenente gli eventi che corrispondono con i filtri di ricerca (possibilmente vuoto)	Risultato atteso
5	Creazione di un nuovo evento con dati validi	Dati dell'evento che si vuole creare, token di autenticazione	Login effettuato con un profilo "organizer"	Viene creato un evento con i dati specificati	Risultato atteso
5.1	Creazione di un nuovo evento senza essere loggati	Dati dell'evento che si vuole creare	Login non effettuato	Viene mostrato un messaggio di errore e l'evento non viene creato	Risultato atteso
5.2	Creazione di un nuovo evento essendo loggati con un profilo non "organizer"	Dati dell'evento che si vuole creare, token di autenticazione		Viene mostrato un messaggio di errore e l'evento non viene creato	Risultato atteso
6	Registrazione di un nuovo utente inserendo tutti i dati	Dati dell'utente che si vuole registrare	Utente non precedentemente registrato	Viene creato un nuovo User nel database	Risultato atteso
6.1	Registrazione di un nuovo utente con dati mancanti	Dati parziali dell'utente che si vuole registrare	Utente non precedentemente registrato	Viene mostrato un messaggio di errore e l'utente non viene registrato	Risultato atteso
6.2	Registrazione di un nuovo utente con email già in uso	Dati dell'utente che si vuole registrare	Utente non precedentemente registrato	Viene mostrato un messaggio di errore e l'utente non viene registrato	Risultato atteso
6.3	Registrazione di un nuovo utente con username già in uso	Dati dell'utente che si vuole registrare	Utente non precedentemente registrato	Viene mostrato un messaggio di errore e l'utente non viene registrato	Risultato atteso
7	Autenticazione con dati corretti	Email e password corretti	Utente precedentemente registrato	L'utente viene loggato nel sistema e può accedere alla propria area personale	Risultato atteso
7.1	Autenticazione con password errata	Email corretta, password errata	Utente precedentemente registrato	Viene mostrato un messaggio di errore e l'utente viene reindirizzato alla pagina di login	Risultato atteso
7.2	Autenticazione con email errata	Email errata, password corretta	Utente precedentemente registrato	Viene mostrato un messaggio di errore e l'utente viene reindirizzato alla pagina di login	Risultato atteso
7.3	Autenticazione senza fornire dati		Utente precedentemente registrato	Viene mostrato un messaggio di errore e l'utente viene reindirizzato alla pagina di login	Risultato atteso

8	Fetching delle location		Location presenti nel database	Viene ritornato un array contenente tutte le location presenti nel database	Risultato atteso
8.1	Fetching delle location senza essere loggati		Utente non autenticato	L'operazione non è permessa, viene ritornato un messaggio di errore	Risultato atteso
8.2	Fetching delle location con un profilo diverso da "organizer"		Utente autenticato con ruolo "user" o "owner"	L'operazione non è permessa, viene ritornato un messaggio di errore	Risultato atteso
9	Fetching di una location in base al nome	Nome della location valido	Database contenente almeno una location con quel nome	Viene ritornato l'oggetto della location corrispondente al nome fornito	Risultato atteso
9.1	Fetching di una location in base al nome senza essere loggati	Nome della location valido	Utente non autenticato	L'operazione non è permessa, viene ritornato un messaggio di errore	Risultato atteso
9.2	Fetching di una location in base al nome con un profilo diverso da "organizer"	Nome della location valido	Utente autenticato con ruolo "user" o "owner"	L'operazione non è permessa, viene ritornato un messaggio di errore	Risultato atteso
10	Creazione di una location	Dati relativi alla location	Utente autenticato con ruolo "owner"	La nuova location viene creata nel database e restituita nella risposta	Risultato atteso
10.1	Creazione di una location senza essere loggati	Dati relativi alla location	Utente non autenticato	L'operazione non è permessa, viene ritornato un messaggio di errore e la location non viene creata	Risultato atteso
10.2	Creazione di una location con un profilo diverso da "owner"	Dati relativi alla location	Utente autenticato con ruolo "user" o "organizer"	L'operazione non è permessa, viene ritornato un messaggio di errore e la location non viene creata	Risultato atteso
10.3	Creazione di una location senza inserire tutti i dati necessari	Dati parziali (e.g., solo nome)	Utente autenticato con ruolo "owner"	Viene ritornato un messaggio di errore e la location non viene creata	Risultato atteso

Table 5: Test case implementati per testare le funzionalità implementate. I dati a cui si fa riferimento nella colonna "Test Data" sono quelli riportati nei Listing 3, 4, e 5.

Di seguito sono riportati per completezza i modelli relativi agli eventi, agli utenti, e ai luoghi.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

const EventSchema = new Schema({
  name: {
    type: String,
    required: true,
    description: "Name of the event"
  },
  description: {
    type: String,
    description: "Description of the event"
  },
  date: {
    type: Date,
    required: true,
    description: "Date and time of the event"
  },
  location: {
    type: String,
    required: true,
    description: "Location of the event"
  },
  latitude: {
    type: Number,
    description: "Latitude of the location"
  },
  longitude: {
    type: Number,
```

```
      description: "Longitude of the location"
    },
    capacity: {
      type: Number,
      required: true,
      description: "Maximum number of participants"
    },
    organizer: {
      type: String,
      ref: 'User',
      required: true,
      description: "Reference to the organizer who created the event"
    },
    pictures: {
      type: [String], // Array of strings for picture URLs
      description: "Array of picture URLs of the event location"
    },
    participants: {
      type: Number,
      default: 0,
      description: "Current number of participants"
    },
    enrolledUsers: {
      type: [Schema.Types.ObjectId], // Array of user IDs
      ref: 'User',
      description: "Array of user IDs representing enrolled users"
    }
  }, {
    timestamps: true
  });

  // Create and export the Event model
  const EventModel = mongoose.model('Event', EventSchema);
  module.exports = EventModel;
```

Listing 3: Modello per gli eventi.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// Define the user schema
const UserSchema = new Schema({
  username: {
    type: String,
    required: true,
    unique: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  role: {
    type: String,
    enum: ['user', 'owner', 'organizer'], // user = basicUser
    required: true
  },

  // Array of saved event IDs
  savedEvents: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Event'
  }],

  // Array of past events (event history)
  pastEvents: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Event'
  }]
}, { timestamps: true });

// Create and export the User model
const UserModel = mongoose.model('User', UserSchema);
module.exports = UserModel;
```

Listing 4: Modello per gli user.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// Define the place schema
const PlaceSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  location: {
    type: String,
    required: true
  },
  capacity: {
    type: Number,
    required: true
  },
  owner: {
    type: String,
    required: true
  }
}, { timestamps: true });

// Create and export the Place model
const PlaceModel = mongoose.model('Place', PlaceSchema);
module.exports = PlaceModel;
```

Listing 5: Modello per i luoghi.

6 Front-End

7 Deployment